

Planning for distributed theorem proving: The teamwork approach

Jörg Denzinger, Martin Kronenburg
Department of Computer Science
University of Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
E-mail: {denzinge, kronburg}@informatik.uni-kl.de

Abstract. We present a new way to use planning in automated theorem proving by means of distribution. To overcome the problem that often subtasks of a problem cannot be detected a priori (which prevents the use of known planning and distribution techniques) we use the teamwork approach: A team of experts independently works on the problem with different heuristics. After a certain amount of time referees judge their results using the impact of the results on the behaviour of the experts. Then a supervisor combines the selected results to a new starting point. The supervisor also selects the experts that will work on the problem in the next round. This selection is a reactive planning task. We outline which information the supervisor can use to fulfill this task and how this information is processed to result in a plan or in revising a plan. Experimental results show that this planning approach for the assignment of experts to a team enables the system to solve many different examples in an acceptable time with the same start configuration and without any intervention by the user.

Keywords: Theorem proving, reactive planning, distributed problem solving

1 Introduction

A major problem of automated theorem proving is the immense search space that even for small problems a theorem prover has to deal with. Research for solutions to this problem centers on two directions, the use of distributed provers and the use of knowledge to guide the prover through the search space. Although this guidance has been used quite some time in provers by means of heuristics, only in the last few years better concepts of guidance, namely planning, have found their way into proving systems. Both, distributed provers and provers using planning have to deal with the same problem: the difficulty, almost inability, of finding a priori appropriate subtasks of a proof problem. Such subtasks can only be determined, if very much about a proof problem is known. But what should be done when the knowledge about the proof problem (besides the input) is vague?

To deal with this problem we developed the teamwork method (see [AD93], [De95]). This method allows the distribution of search processes where the descriptions of the processes show no obvious ways for distribution. The general

idea of teamwork is to let a team of several so-called experts (which are computational agents) work independently on the proof problem. They differ in the heuristics they use to determine the next step to do. After a given amount of time the experts stop their work and for each expert a referee assesses the work of the expert and reports a general assessment and a few good results to a supervisor. This supervisor collects the reports, generates – based on these reports – as a new starting point for the experts a new set of equations, and selects experts and referees for the next working round. We call the phase when the referees and the supervisor are working a team meeting.

Structure and behaviour of our teams lead to a system that uses both, competition and cooperation between its components (mainly the experts) to solve given problems. The experts compete with each other in order to stay in the team. But with the help of the referees they also cooperate, because their best results are used to form a new starting point for the work of the team.

In [AD93] we concentrated on the general aspects of this approach and showed that for many examples there exist teams that allow enormous speed-ups. In [DF94] we designed powerful experts and demonstrated that we are able to solve examples using teamwork that could not be solved by any of our experts working alone. But the question remains how to choose *automatically* good teams for an example, so that a lot of different examples can be solved in an acceptable time without any intervention by the user. The solution to this problem is the supervisor. During the team meetings it selects the members of the next team.

We see this process of creating a new team in each team meeting as a kind of *reactive planning*. The supervisor uses a long-term memory (general knowledge about experts, referees and their relationships, dependencies and incompatibilities) in combination with a short-term memory (the results of the experts on the given problem so far) to determine the team members. It not only selects the team members of the next round, but also can make assignments for further rounds provided that the selected team comes up with the expected results. Otherwise the behaviour of the selected team is used to find better suited experts. This is a kind of replanning or plan revision which we call *maintenance* of a plan.

For planning a team in this way it is not necessary to find subtasks of a proof problem. If subtasks can be detected, they can be assigned to experts that are capable to prove them. If no subtasks can be found, the supervisor tests several experts and adjusts the whole system more and more to the given problem.

Using reactive planning by the supervisor we were able to solve automatically most of the examples of [DF94] and [AD93] without any intervention by the user: no team selection, no parameter adjustment by the user was necessary, just planning using general but vague knowledge was enough (see section 5). The knowledge-based reactive planning approach used by the supervisor of a team presented in this paper allows for a much easier extension of the system to new domains than the auto mode of Otter (see [Mc94]) does. There the code must be changed in order to deal with new domains. This auto mode was, like teamwork, designed to provide a fully automatic theorem prover to users who do not want to learn all about the tricks of a prover in order to use it successfully.

2 Automated Theorem Proving and Completion

Since this paper is aiming to analyze and describe the planning aspect of teamwork, we only give a very brief introduction into automated theorem proving, equational theorem proving and the completion method for equational proving. For more details we refer to [CL73], [HR87], [BDP89] and [AD93].

Theorem proving means solving the following problem:

Given: A set A of axioms and a theorem T to prove.

Question: Is T a logical consequence of A ?

In equational theorem proving $A = \{s_i = t_i \mid i = 1, \dots, n\}$ is a set of (universally quantified) equations and T is an equation $u = v$, too.

All successful methods for automated theorem proving, if equality is involved, are based on two kinds of inference rules: generation rules and contraction rules. The generation inference rules add new facts to the data base. These facts are derived either from the axioms alone (as in the case of equational theorem proving by completion, i.e. the critical pair generation) or from both the axioms and the theorem T (as in the case of resolution and paramodulation). The contraction inference rules change or delete facts from the data base. A well known contraction inference is the reduction (or demodulation) that uses an equation $l = r$ as a rule $l \rightarrow r$ in order to exchange instances of l in a fact by the appropriate (smaller) instance of r . A fact is in normal form, if no reduction with any equation of the data base is possible.

From a theoretical point of view we need for a theorem prover, besides the inference rules, fairness criteria for the use of these rules. These criteria guarantee that each application of an inference rule, that is enabled infinitely often, will finally be executed. But, for challenging examples there are many possible inferences and a systematic application results both in enormous run times and the need of much (memory) space. For such examples the prover very often has an agenda of 50,000, 100,000 or even more inference rule applications.

Therefore, implementations of automated theorem provers use strategies and heuristics to select the next inference rule to apply and the facts these rules should work on. A strategy guarantees theoretical completeness whereas for heuristics it is possible that the prover will not find a proof even if there is one and enough time and space are provided. Many theorem provers allow the user to choose between various strategies and heuristics. But two problems remain. First, for a given problem there may be no appropriate strategy or heuristic implemented in the system. Second, even if there is a good one implemented, how does the user know which one is good? We tackled both problems with our teamwork method as we will demonstrate in the next section.

3 The Teamwork Method

The teamwork method is inspired by project teams in business companies. Due to their temporary existence they allow an exact tuning to the problem they have to solve. This tuning is achieved by the supervisor of the team who always

chooses the team he thinks is best suited for the current status of the solution of the given problem.

So, one major task of the supervisor is a planned selection of the team members. This planning must also allow a fast reaction to problems concerning the selected teams or when unexpected breakthroughs occur. We will discuss this task of the supervisor in more detail in the next section. In this section we will look briefly at the other components of a team, the experts and referees, and their tasks and how they compete and cooperate to solve a given problem.

Experts are those components that work on the problem solution by generating new facts. Each expert has to work on an own processor. Usually there are more experts than processors. Therefore the experts have to compete with each other to get a processor for the next working period. Each expert uses other methods to gain new data. In automated theorem proving the experts differ in the selection methods for the next inference step. There are many criteria that can be used to get different heuristics. For example, one can focus on parts of the set of known facts, focus on statistical properties of the facts (for example the number of symbols), or focus on the goal to prove (see [AD93], [DF94]).

Experts compete for processors, but another characteristic of a project team is that the members of the team cooperate with each other. As a result of this cooperation, problems can be solved much faster than by a single team member alone, even if we consider as the time for finding the solution the sum of the times needed by all members. So, cooperation leads to *synergetic* effects.

The teamwork method achieves cooperation between the experts by using referees and team meetings. After certain periods of time during which the experts work independently on the problem, a team meeting takes place. Before the meeting each expert and its work is evaluated by a referee. The tasks of the referees are to compute a measure for the success of their experts and to select good results of the experts. The measure of an expert is used by the supervisor to determine the team members of the next round. The selected results are used to generate a new data base of facts that is the starting point for the next round.

The main problem in designing referees is how to compute measures for the overall behaviour of an expert on a problem and for the usefulness of generated results. For both measures there are statistical criteria that have proven to be quite successful in our application, namely automated theorem proving, and which can be computed efficiently. Our referees use a weighted sum containing, besides others, the following numbers: the size of the data base, the number of contraction inferences that were performed, and the number of potential inferences, which are the critical pairs in our case (again, see [AD93]). Referees differ in the weights they use to compute the weighted sum.

In order to select good results we use similar statistical criteria, but compute them for each fact, for example the number of contractions performed with this fact. Moreover, we give the referees a maximal number of facts that can be selected and a minimal measure that each selected fact must achieve. Thus we avoid blowing up the search space with unnecessary results.

The supervisor constructs the new starting data base by using the data base

of the best assessed expert as basis. This guarantees completeness of our prover. Then it adds the selected results of the other experts to this basis. This combination leads to a significant speed-up in most examples.

4 Planning by the Supervisor

As in the case of human project teams, systems based on the teamwork method rely on a good use of the given resources, i.e. the processors. The supervisor is responsible for the assignment of these resources. In order to find such an assignment it has to use planning. The main problem one has to face if one wants to use planning in theorem proving is a *general vagueness of all information* one can use. In contrast to planning in the blocks world or even the navigation of robots, there are no operators with fixed pre- and postconditions that can be put together in order to obtain an executable plan. In theorem proving the only candidates for such operators are the inference rules themselves and using them would lead back to the initial proof problem.

Therefore one has to do the planning on an abstract level. But this can lead to two new problems. On the one hand it may be possible that – due to the abstraction – there is not enough information to generate a plan on the abstract level at all. On the other hand it could be possible that the next step of a generated abstract plan cannot be executed on the concrete level of the inference rules, because prior steps did not deliver the expected results. In both cases one can say that a reliable plan can be constructed more frequently if more knowledge about the current problem is available. This is the way the known planning approaches for theorem proving, for example [Bu88] (see section 6), deal with this problem. They require a lot of very concrete, even specialized knowledge – for example detected subproblems – to be able to operate.

The abstraction level provided by the experts and referees in the teamwork method is even higher. This means that our planning approach uses very vague information and that it especially does not rely on the detection of subproblems. The consequences of planning on such a high level are: Firstly, we always have to expect that generated plans are vague and that they may be wrong. Secondly, at no time we can plan the complete proof, because the more future steps we plan the more vague is the information about the outcome of these steps.

Since we use the teamwork method to handle the problem of vagueness we have to deal with a second problem: the supervisor which is responsible for the planning is the *bottleneck of our distributed system*. Therefore we have to limit the time the supervisor has for planning to a certain period of time, which must be adapted to the current state of the plan and the proof and which must be small with regard to the length of a working period.

Our solutions to these problems are based on three concepts. The fundamental concept is *teamwork* itself. Since the experts make it possible to follow several promising directions at once some vagueness can be compensated. The referees enable the system to find the most promising direction (expert) and good results of other experts. The reports of the referees also help the supervisor to decide

whether a plan must be changed or not. This leads to our second concept, the general planning approach, which is based on the ideas of *reactive planning* (see [Mc90], [Be91]). The main idea is to combine a long-term memory with knowledge about earlier proof attempts and a short-term memory containing all the information about the experts and referees concerning the actual problem. By combining these two kinds of knowledge the supervisor can easily react to unsuccessful plans and perform replanning. The third concept solves the “bottleneck problem”. We implemented the planning of the supervisor as an *anytime algorithm*, which means that the planning can be stopped at any time and a plan will be available. We hope that a plan is improved if the planning time increases but due to the vagueness of the underlying knowledge this cannot be guaranteed.

The following two subsections describe these three concepts and the planning process in more detail. In 4.1 we introduce the notion of a *domain* and show how a domain can support the planning process. Subsection 4.2 deals with the *maintenance* of the plan performed by the supervisor during the team meetings.

4.1 Domains

A domain is – as in other proof planning systems – a collection of information about a set of proof examples one is (or was) interested in. In our system a domain consists of facts defining the domain, consequences of these facts, methods suitable to the domain and further informations. The information about a domain is represented in a frame. The following example shows the description of the domain ring. Such domain frames and the expert frames, which are introduced in the following section, are constructed (at the moment) manually by exploiting the experience made by several experiments.

domainname:	ring
signature:	+ :2; 0:0; -:1; *:2
equations:	$x+0 = x$; $x+(-x) = 0$; $(x+y)+z = x+(y+z)$ $x+y = y+x$; $(x*y)+(z*y) = (x+z)*y$ $(x*y)*z = x*(y*z)$; $(x*y)+(x*z) = x*(y+z)$
consequences:	$0+x = x$; $(-x)+x = 0$
starting team:	Add-Weight; Add-RWeight
middle team:	Add-Weight; Add-RWeight
end team:	Add-Weight; Goal-in-CP
superior domain:	group
specialized domains:	boolean ring
similar domains:	non-associative ring

Detection of a Domain. Before the supervisor can use any information about a domain for its planning the domain must be detected in the current proof problem. This detection process is performed by an expert, a so-called *domain detection specialist*. This specialist is always a member of the first team. Note, that by using an expert for this task we avoid consuming the time of the supervisor, our bottleneck. Moreover, parallel to the detection process, other experts can already be trying to solve the example. The specialist checks many domain descriptions and reports to the supervisor those domains for which the detection

process was successful. In general, several domains are found. It may also report the equations of the slot **consequences** (or a referee's selection of them).

There are several possibilities to determine whether an example belongs to a domain or not. They differ in the amount of deduction used and therefore in the chances that a domain is detected. They all have in common that they try to find a match Σ from the signature of the domain (slot **signature**) to the signature of the example and then perform tests with the facts listed under **equations**, instantiated by Σ . If the tests are successful, the domain is detected.

The first possibility is to simply test, if all instantiated equations are among the facts of the example (up to renaming of variables). In practice this method cannot deal with situations in which equations from the example can reduce other equations that are needed for the detection. In such situations the domain is not detected although the example belongs to the domain.

The second possibility is the other extreme: for all facts $s=t$ in **equations**, it is tested, if $\Sigma(s) = \Sigma(t)$ can be proved as consequence of the set E defining the example. In this case we have to show many equations instead of one with a process that may not terminate, if an equation is not a consequence.

The possibility we have chosen lies between these extremes. For each equation $s=t$ in **equations** we check, if the normal forms of $\Sigma(s)$ and $\Sigma(t)$ are identical or if $\Sigma(s) = \Sigma(t)$ is subsumed by an equation of the example. This overcomes the problem of the first possibility while still being a decidable (and fast) test.

Determination of a First Plan. In the first team meeting the supervisor begins with the planning of the team. Because the short-term memory consists only of the referee reports of one working period, the composition of the next team is determined by the information provided by the long-term memory. A central part is played by the *plan skeleton* of the "best" detected domain. A plan skeleton (to a domain) is represented by the contents of the slots **starting team**, **middle team** and **end team** of a domain. The experts in these slots have proven to be useful for many examples of the domain in the three phases we have observed in many proofs. If the supervisor determines that a proof attempt has reached the middle or the end phase, then the experts mentioned in those slots should be used in the next team. Note that there are powerful experts that have been designed for the use in the end phase of a proof (see [DF94]).

With the selection of the "best" domain of an example we have the core of the next team. For determining this domain, the supervisor uses the hierarchical information that is given in the slots **superior domain**, **specialized domains** and **similar domains** of the domain description. By eliminating all those detected domains that are superior domains to another detected one or that are similar to a domain which is superior to a detected domain we can obtain a hopefully small set of candidates or even a single one. Since in the first team meeting the supervisor has no information about these remaining domains with regard to the actual proof problem a random candidate is chosen as the domain the supervisor will concentrate on (for the moment).

If there are more processors available than there are experts in the starting team of the chosen domain, then the domain detection specialist can be incor-

porated in the team again with the task to look especially at those domains that are specialized versions of the chosen one. Additionally, experts that are mentioned in the starting teams of other detected domains can be used to complete the team.

4.2 Maintenance of the Plan

As already stated, one can never be sure that a plan or a plan skeleton succeeds in finding a proof of a given problem. In most cases adjustments of the plan have to be made, ranging from small changes up to generating a totally new plan. This *maintenance* of the plan is the task of the supervisor and is performed in four steps during a team meeting: restricting the time for planning, determining proof phase and domain changes, selecting the next team, and computing the next cycle length.

Determination of the Time Available for Planning. As already stated, the periods of time in which the supervisor is active have to be very short. Therefore we allow the supervisor to use 1% of the length of the last working period.

Change of the Proof Phase and Selection of a new Domain. We define the start of a new proof phase as the team meeting in which most members of the suggested team of the prior phase either are not members of the actual team anymore or have a measure that is below a predefined percentage of the best expert's. If the start of the next proof phase is detected, then the members in the appropriate slot of the chosen domain will be members of the next team.

There are two reasons for changing the chosen domain and consequently the plan skeleton that is used. The first reason is that the specialist detects a new domain which is a specialization of the already used one. Then the supervisor immediately switches to the plan skeleton of this new domain. If a new domain is detected that is superior to the chosen domain, then no modification of the plan is necessary. The second reason is that the experts of the chosen plan skeleton are much worse rated by their referees than the best experts of the last team (and a change of the proof phase does not alter this fact).

If this second reason occurs all domains that have been detected so far are rated as follows: For each domain the supervisor sums up the measures of the experts which are mentioned in the plan skeleton of the domains. Each measure of an expert is weighted by 1 divided by the number of working periods since the measure was given. In this way older information gets less credit. The domain with the best sum is selected, if a certain threshold is reached. Otherwise the supervisor uses no domain information to choose experts. If an expert has never been member of a team during the proof attempt, it gets a measure of 0.

Selection of the Members of the next Team. Since the competition aspect is an important feature of teamwork, the supervisor always selects the best expert of the last working period as member of the next team. Also those experts that are suggested by the plan skeleton and have not performed much worse than the best expert get a place in the next team. The remaining places in the team are filled according to the following routine.

If the plan skeleton of a new domain has been selected or the proof phase has changed the experts mentioned in the corresponding slot of the actual plan skeleton get places in the next team, provided they did not perform unsatisfactorily in the last team. If there remain open places, then some computation is necessary to decide which experts should take these places. This computation is designed and implemented as an anytime algorithm, which – as already stated – means that the algorithm can be stopped at any time and the open places can be filled as good as possible with regard to the time used for this computation. To achieve this we want to examine at each time only a few experts and this examination is done in several rounds taking more and more criteria into account. In order to have always only a few experts under consideration we group the experts into five classes which are in order of importance:

- the experts of the last team,
- the experts that work well together with the best expert of the last period,
- the experts that work well together with the experts of the current phase of the plan skeleton,
- the experts that are suggested by other detected domains,
- all other experts.

For doing this classification and also for the examination and evaluation of the experts we need further information about experts. Like the domains we represent the knowledge about an expert in a frame. The following is an example.

```

expertname:      Add-Weight
robustness:    0.8
knowledge involved: 0.1
proof phase:   start: 0.6; middle: 0.5; end: 0.5
referees:      start: Statistic-1; middle: Statistic-4; end: Statistic-6
domains:       all
similar experts: Add-FWeight-1; Add-RWeight
cooperative experts: Goal-in-CP; CP-in-Goal
impossible experts: none

```

The experts of the slot **cooperative experts** form the group of experts that work well together with the expert described in the frame (in our example Add-Weight). The other information is needed to determine a weight for this expert.

For each expert of the actual group the computation of this weight is done several times, each time taking more criteria into account. These criteria are:

- How good was the expert in the last phase?
- How good is the rating the expert received with respect to the detected domains and the phase of the proof?
- How good were the results of the expert in earlier phases of this proof?
- How well does the expert cooperate with the already chosen team members?
- How specialized is the expert?
- How good is the robustness of the expert?

Each criterion will lead to a measure between -1 (bad) and 1 (good) and the weight of the expert is a weighted sum of these measures. For each time the weight is computed, adjusted weights for the measures have to be used in order to compare experts of groups that were fully evaluated with experts of a group

for which only weights using part of the criteria could be computed (due to lack of time). For comparisons always the weight using the most criteria is used. Also for experts that never have been team members or that have not been members of the last team adjusted weights have to be used.

If we have n free processors, the supervisor will choose the n experts with the highest weight. Note that the more time the supervisor has the more experts can be examined and the more knowledge can be used to come to a decision. But, because of the vagueness of the information we use, there is no guarantee that this decision will really improve over the time. Let us now take a closer look at the criteria we use to examine and evaluate experts.

The rating of the suitability of an expert for a domain takes into account, whether the expert is member of the team of the plan skeleton of the domain for the current phase and whether the domain is in the **domain** slot of the expert. If this is the case for all detected domains we would get a measure of 1. Note that experts that are not members of a plan skeleton of a domain may have the domain in their **domain** slot.

The measure that represents the history of the expert on the current proof attempt is computed as the mean value of the comparisons of the expert with the best experts of the working phases when the expert was member of the team. We get the comparison by dividing the result of the expert by the result of the best expert.

The measure for cooperation uses the slots **similar experts**, **cooperative experts** and **impossible experts**. For each already chosen expert we add 1, if the chosen expert is in **cooperative experts**, we add -1, if the chosen expert is in **impossible experts** and we add -0.1, if the chosen expert is similar to the expert we check at the moment. We add a small negative number, because this way the more similar experts we have in a team the more unlikely it would be to add another similar one. Then the sum is divided by the number of chosen experts that are mentioned in the three slots.

If at least one domain was detected, we use the value of **knowledge involved** as indication for the specialization of the expert. Finally, we multiply **robustness** with the appropriate value of **proof phase** to get a measure of the robustness of the expert.

Determination of the Length of the next Working Period. It is quite obvious that the length of the next period has to take into account the members of the new team and especially the differences between new and old team. Therefore, if the plan has been successful so far, meaning that the experts of the chosen plan skeleton or all experts of the last round have had good measures, then the lengths of the working periods increase linearly. When most of the experts did not have the time to perform more than 10 inference steps, the supervisor will use an exponential growing length.

If most of the experts of the next team are new, then it is difficult to tell whether the team will be good or bad. Therefore the length of the next working period will be shorter. How short depends on the number of facts that constitute the current problem description. The reason for this is that the more facts there

are the more time is needed to perform an inference step and that, in order to get useful measures from the referees, the experts have to perform several inference steps in the working period. If the team was successful, then it will get more time the next round, else other experts will be tried.

5 Experiences

In the last section we described how the supervisor plans a proof attempt and reacts to the reports that it gets from the referees. In this section we will demonstrate that this planning enhances the performance of the whole system.

Information about the implementation of the system can be found in [AD93]. For a description of the strategies and heuristics of the experts see [DF94].

In [AD93] and [DF94] we showed by experiments that teamwork, without planning by the supervisor, can reduce the run-time of a system on a proof problem dramatically compared to the run times of the used experts when working alone. But these results have a drawback: *we* selected the team members of the teams. And these teams changed from example to example.

It is well known that all automated theorem provers have many parameters that can (and must) be adjusted to a given proof problem. Our important parameter was the composition of the team. But this requires that the user of a theorem prover has much knowledge about the prover and its parameters, so that he can plan his proof attempts. But we want a system which can solve many, very different examples in an acceptable time without any help of the user.

We demonstrate that teamwork with planning enables us to build such a system by reporting results obtained with examples from four domains: propositional calculus (cal1 to cal4), lattice ordered groups (pla to p10), boolean rings (bool5b) and rings (lusk6). For the descriptions of these examples see [DF94] and [AD93]. We added the last two examples to make the detection of domains more difficult. The four domains provide a wide range of equational problems.

Table 1 documents our results. Besides the run-times of a team using planning we give the run-times of the best team consisting of two experts we were able to find, the two experts that form this best team and the run-time of the best expert working alone on the problem. We have chosen to use only two processors because with small resources a good use of them is important. In order to allow a better comparison we also restricted the best user selected teams to two processors.

The team runs using planning were always started with the same starting team and the same system configuration. Besides the input equations, a reduction ordering, the goal of an example and the start command, no interaction between system and user took place. The alterations in the composition of the team were only effected by the actions described in section 4. The system configuration specially includes some basic data for computing the lengths of the working periods which play an important part in the run of a proof. Note that for the times of the best teams the lengths of the several periods have also been adapted by the user to the specific examples.

The main observation in Table 1 is that the team with planning needs more time than the best team for most of the examples but still can also solve those

example	team with planning	best team	best team members	best sequential expert
cal1 (luka1)	29.91	35.07	MaxWeight, CP-in-Goal	40.99
cal2 (luka2)	50.18	14.21	AddFWeight, GTWeight	45.02
cal3 (luka3)	84.86	96.47	Goal-in-CP, CP-in-Goal	—
cal4 (luka4)	202.89	72.19	Goal-in-CP, AddRWeight	297.16
p1a	0.71	0.28	Occnest, MaxWeight	0.27
p1b	0.68	0.47	Occnest, AddWeight	0.28
p2a	2.46	5.41	AddRWeight, Occnest	79.52
p3a	3.04	4.23	Occnest, AddWeight	4.14
p3b	2.94	2.62	GTWeight, Occnest	2.55
p4a	2.02	2.46	GTWeight, Occnest	1.84
p4b	1.93	2.06	Occnest, AddWeight	1.71
p6a	0.84	0.40	Occnest, MaxWeight	0.39
p6b	0.58	0.16	Occnest, MaxWeight	0.16
p8b	93.54	56.84	MaxRWeight, Goal-in-CP	—
p9a	22.58	8.66	Occnest, AddWeight	19.57
p9b	23.95	8.44	AddWeight, Occnest	50.95
p10	37.94	25.20	MaxRWeight, Goal-in-CP	—
bool5b	50.11	58.86	Goal-in-CP, AddWeight	—
lusk6	500.08	307.96	AddWeight, AddRWeight	3019.00

Table 1: comparison team with planning vs best team and best expert (in sec)

examples that no single expert can solve. It is clear that using planning we have to expect a certain overhead. Our analysis of the runs using our proof extraction and analysis tool (see [DS96]) showed that not the time for doing the planning is responsible for the longer run times but the need to try out experts and the replanning that is involved when adjusting the team to a problem.

If we take a look at the experts of the best teams, it is quite obvious that even in one domain very different teams were needed. Especially in the domain propositional calculus for each example a different team was best. Therefore it cannot be expected that the first plan to a domain always succeeds. Instead the reactive part of the system must detect and exchange bad experts.

Interestingly, there are some examples (cal1, cal3, bool5b, p2a, p3a, p4a, p4b) for which the team with planning needs less time than the best two-expert team. While the run times for the lattice examples are so short that this would not be significant, the other three examples proved to be interesting. Our analysis (and later experiments) showed that the better run times of the team with planning were due to the fact that experts that were not members of the best team – but chosen by the supervisor in the run using planning – provided results necessary for the proof a little earlier than the members of the best team. But they were not able to produce enough results to form a better two-expert team with one of the members of the best team. Using a team with three experts we were able to obtain a better run time than the team with planning. For these examples planning allows us a better use of the resources. But in general we have to expect

that a change of the initial plan is necessary for many examples and that in some working periods some experts do not contribute to a proof.

If we compare the team using planning with the best sequential experts for an example we can observe that for small, easy examples the best sequential expert finds a proof faster. But for harder examples the team with planning clearly outperforms the best sequential expert and it still finds proofs when no sequential expert can. If we would compare our team using planning with a fixed sequential expert there would be much more examples for which this expert would find no proof. So the synergetic effect of teamwork can also be observed when the team uses planning.

Finally we have to point out that in section 4 two ways of using a detected domain have been described: for adding known consequences and for planning purposes of the supervisor. In all the examples listed in Table 1 a domain was only used in the second way, in order to emphasize the planning aspect of a domain.

6 Related Work

The work presented here is related to three areas of artificial intelligence, namely automated theorem proving, planning and distributed artificial intelligence. The first work that is related to the first two areas is due to A. Bundy (see [Bu88]), who coined the term *proof planning*. He concentrated on inductive theorem proving and used a STRIPS-like (see [FHN81]) planning approach. He invented so-called tactics that are similar to the operators that can be defined in STRIPS. A proof attempt consists of two phases, a planning phase, where on a meta-level a plan is constructed using the tactics, and a proof phase, where the selected tactics are evaluated on the level of inference rules.

The problem of this approach is that the domains of the proof problems have to be understood very well, so that it will be possible to find appropriate subproblems to a proof problem. In equational or first-order theorem proving this is not the case as we stated in the introduction. The information about domains we have access to is much too vague to allow the use of Bundy's approach.

In the area of planning we were inspired by the works of McDermott and Beetz on planning reactive behaviour (see [Mc90], [Be91]). Here planning was intended to help a robot navigate through an area and perform certain tasks with limited planning time. This limitation is also an important point of our approach. Although robot control and automated theorem proving are very different areas, both, as stated before, have to deal with vague information. By the use of several experts we have the possibility to choose the situation we want to continue on, which is not possible for only one robot.

In the areas of planning and distributed AI the research mainly concentrates on planning for autonomous agents, i.e. systems without a central control (see for example [DL87]), or on central planning of tasks that require coordination, because there are dependencies between the actions of the agents (for example plans for several robots, see [Ro82]). As the supervisor is the central control of the team and theorem proving using teamwork is a task where no dependencies occur, we have easy solutions to most of the problems addressed in these papers.

7 Conclusion and Future Work

We have presented a distributed theorem proving method where planning of the assignment of agents to the processors allows us to improve significantly the number of theorems that can be proved without the user fiddling with parameters of the prover. Although the run times of the version of our prover using planning are slower than the run times of the best known teams for the problems we were able to prove examples from different domains to which none of our sequential provers could find a solution thus still showing synergetic effects. Further, reactive planning enables us to prove examples from one domain where the best known teams differ from example to example.

Our approach to proof planning allows us to deal with knowledge about domains that is vague, unstructured and possibly contradictory. This is due to the competition of the experts in the teams. All other approaches to proof planning require exact and often total knowledge about a domain of interest in order to achieve satisfactory results. Furthermore, the addition of new domains to our system is easy because of the explicit representation of the knowledge by frames.

The detection of subproblems and the use of special methods to solve them – which are characteristic for other approaches – can also be integrated in our approach without losing the ability to deal with vague information. This is one direction into which we want to investigate. Other topics of future research are to automate the generation of domain information by learning from examples and the improvement of planning by not only selecting known experts but also by generating new experts using parameter adjustments of generic experts.

References

- [AD93] **Avenhaus, J.; Denzinger, J.:** *Distributing equational theorem proving*, Proc. 5th RTA, Montreal, LNCS 690, 1993, pp. 62-76.
- [BD88] **Boddy, M.; Dean, T.:** *An Analysis of Time-Dependent Planning*, Proc. 7. National Conf. on AI, Minneapolis, 1988, pp. 49-54.
- [BDP89] **Bachmair, L.; Dershowitz, N.; Plaisted, D.A.:** *Completion without Failure*, Coll. on the Resolution of Equations in Algebraic Structures, Austin (1987), Academic Press, 1989.
- [Be91] **Beetz, M.:** *Decision-theoretic Transformational Planning*, Internal report, Yale University, 1991.
- [Bu88] **Bundy, A.:** *The use of explicit plans to guide inductive proofs*, Proc. 9th CADE, 1988.
- [CL73] **Chang, C.L.; Lee, R.C.:** *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.
- [De95] **Denzinger, J.:** *Knowledge-Based Distributed Search Using Teamwork*, Proc. ICMAS-95, San Francisco, AAAI-Press, 1995, pp. 81-88.
- [DF94] **Denzinger, J.; Fuchs, M.:** *Goal oriented equational theorem proving using teamwork*, Proc. KI-94, Saarbrücken, LNAI 861, 1994, pp. 343-354.
- [DL87] **Durfee, E.H.; Lesser, V.R.:** *Using Partial Global Plans to Coordinate Distributed Problem Solvers*, Proc. IJCAI-87, 1987, pp.875-883.
- [DS96] **Denzinger, J.; Schulz, S.:** *Recording and Analyzing Knowledge-Based Distributed Deduction Processes*, to appear in Journal of Symbolic Computation, 1996.
- [FHN81] **Fikes, R.E.; Hart, P.E.; Nilsson, N.J.:** *Learning and executing generalized robot plans*, in Webber, Nilsson (eds.) Readings in AI, 1981, pp.231-249.
- [HR87] **Hsiang, J.; Rusinowitch, M.:** *On word problems in equational theories*, Proc. 14th ICALP, Karlsruhe, LNCS 267, 1987, pp. 54-71.
- [Mc94] **McCune, W.W.:** *OTTER 3.0 Reference manual and Guide*, Tech. Rep. ANL-94/6, Argonne National Laboratory, 1994.
- [Mc90] **Mc Dermott, D.:** *Planning reactive behaviour: A progress report*, in J. Allen, J.Handler, A. Tate: Innovative Approaches to Planning, Scheduling and Control, Kaufmann, 1990, pp.450-458.
- [Ro82] **Rosenschein, J.S.:** *Synchronization of Multi-Agent Plans*, Proc. AAAI-82, 1982, pp.115-119.